Distributed Agile

By Dr. Christoph Steindl, Senior IT Architect and Method Exponent,
christoph_steindl@at.ibm.com

Agile Software Development (see [Agile Manifesto]) is a relatively new trend with special flavours (such as Extreme Programming [Beck], Scrum [Beedle, Schwaber], Crystal [Cockburn], Feature-Driven Development [Palmer], Dynamic Systems Development Method [DSDM] etc.). Iterative and incremental development, however, has long been around (see [Larman1], [Larman2]).
With the open source movement, **distributed** development has become widespread and apparently successful.
The following short article is about a combination of both, called "**Distributed Agile**". It points to challenges of distributedness and countermeasures ("*Therefore*:") against those challenges in a concise form (for more background see [Simmons], [Fowler], [Steindl]).

Challenges of distributedness and countermeasures

Decreased communication bandwidth

- Different time zones
  - If you're lucky, you have a couple of overlapping work hours.
  - Because of the delay for replies you may assume the obvious (which may be plain wrong) instead of asking.
- Cost and quality of telecommunication
  - Sometimes you may not even get a free telephone line to remote sites.
  - Access to a common repository can be VERY slow.
  - Bidirectional communication (with VoIP, builtin microphone and speakers) may lead to acoustic feedback.
    Therefore: Use headsets or ear plugs to avoid acoustic feedback.
  - Screen sharing may be very slow, tools (like NetMeeting) may not work reliably or become very slow with more than 2 members in a session.
    Therefore: Expect a lot of troubles with the technical infrastructure for collaborating across continents.
- Don't expect the developers to read the documents / work products.
  - Don't rely on written documents, but on direct communication.
    Therefore: Talk a lot, even about obvious things, in addition to daily status meetings.
    Therefore: Get early feedback from the customer (with visual prototypes, halfway through the iteration) for early validation (and not just verification).

Therefore: Use direct communication and instant messaging rather than emails to:

- Avoid the time lag.
- Avoid the misunderstanding.
- Reach consensus, check whether the other parties really have understood what you mean, check again and again.

Therefore: Use Collaborative Tools (Wiki, FitNesse, Blog, SameTime, NetMeeting, Mailing List, Forum, Web Cam, Telephone).
Therefore: Use Distributed Pair Programming (e.g. Pair Programming with screen sharing and VoIP) [Williams].
Therefore: Hold distributed iteration planning meetings ("Sprint Planning Meetings" in Scrum) and iteration demos ("End-Sprint Demos" in Scrum) with the customer.
Therefore: Have short daily status meetings ("Daily Scrums").
Therefore: Use automated tests to specify the requirements:

- Written specifications can hide details (especially between the lines ;-)
- With every hand-over of a written document, the tacit knowledge of the author (typically 80% are tacit!) is left behind.
- Test cases make the details and the flow of events (dynamics) explicit.

Therefore: Use Test-Driven Development at the acceptance-test level (i.e. customer develops acceptance tests at the start of an iteration, developers are driven by the acceptance tests).
Therefore: Separate teams by functionality (each team develops an entire module/subsystem/component…) not activity (one team for analysis & design, one team for development, one team for maintenance,…)
Therefore: Hold project retrospectives at the end of each iteration (at least at the end of each release) to identify what worked well, what didn't work well and what to try [Kerth].
Therefore: Build a common understanding by using an ubiquitous language and enforcing Domain-Driven Design [Evans].

Decreased visibility into project status

- Difficult for project managers, customer and business sponsors to measure and see the progress
- Different interpretations of "xx% complete"
  Therefore: Use short iterations (rather shorter than longer).
  Therefore: Show prototypes to users (even during an iteration, "User Viewings" in Crystal).
  Therefore: Deliver software frequently (after every iteration or every other iteration)
  Therefore: Build up common understanding of what is necessary for shipping software
    - What kinds of tests
    - What levels of quality
    - What kind and amount of documentation

  Therefore: Have customer participate in daily status meetings.
  Therefore: Show progress charts ("Burndown Charts" in Scrum) publicly.
  Therefore: Build a tight safety net with unit tests ([Rainsberger], [Massol], [Hunt]).

Therefore: Use Test-Driven Development at the acceptance-test level (i.e. customer can execute the tests during the iteration to see what already works).

Problems with configuration management

- Troubles when integrating pieces from different teams.
- Troubles with different development environments.
- Troubles with replicating the customer's system environment on the various development sites.
  Therefore: Use common tools:
  - Integrated development environment (e.g. eclipse)
  - Source Code Repository (e.g. CVS), [Thomas]
  - Build environment (e.g. Maven)
  - Unit test framework (e.g. Junit)
  - Acceptance test framework (e.g. FitNesse)
  - Automated build server (e.g. CruiseControl)
  - Bug tracking system (e.g. Bugzilla)

  Therefore: Integrate early and often
  Therefore: Automate builds [Clark]
  Therefore: Use Continuous Integration

Disconnection on project estimation

- Different experience and background of people who estimate the work and perform the work.
- Little identification with estimate.
- Difficult to assess the accuracy or reasonability of estimates as the project progresses.
  Therefore: Have the developers select and estimate their tasks.
  Therefore: Have the developers update their estimates regularly (e.g. every day or other day).
  Therefore: Display status / effort remaining publicly, use information radiators.
  Therefore: Measure team velocity (how much functionality the team can ship in an iteration).

Cultural Differences and Teaming

- Different value systems, education systems, political systems etc.
  Therefore: Send Ambassadors.
  Therefore: Allow several weeks for getting used to self-organization:
  - If you're used to "command & control", you want detailed work lists, you don't estimate your work as a developer, you want to be left alone for some time, you don't want to give others visibility into your way of working
  - Command and control hides a lot of things and fosters the "blame game". If something is not how you like it to be, you just do whatever you like and blame someone else.
- Different intrinsic motivation.
- Expect a lot of difficulty understanding each other:
  - English may be nobody's native tongue.
  - Refusing requests ("saying NO") may be very unnatural in some cultures, politeness can hide misunderstanding.

  - Instant messaging (e.g. with SameTime) has several advantages over oral communication:
    - It’s easier to understand written English.
    - You can store the transcript and forward it to those who couldn’t attend.
- Expect high staff turnaround
  - Job hopping has been a well-known issue in the Silicon Valley, the job market is over-heated in India.
- Quality standards are harder to reach
  - In a co-located team, you can reach quality standards by pair programming, coding standards. Knowledge diffuses easily by osmotic communication, you can build shared values just by walking around and talking.
  - In a distributed setting it’s harder:
    - less identification with the project (the project is not "your baby")
    - less identification with the employer (gold-rush environment, where you leave immediately if you're paid more)
    - less responsibility (devil-may-care)
    - less teamwork (you do what you're told)
    - easier to get away with not following the rules

  Therefore: Use automated tools to detect problems early (Clover, JCoverage, checkstyle, copy&paste detector of the Maven plugin PMD,…)

References

References to tools (NetMeeting, Wiki, FitNesse, SameTime, eclipse, CVS, Maven, Junit, CruiseControl, Bugzilla, Clover, JCoverage, checkstyle, Maven plugin PMD) have not been given since they are easily found with Google: just type in the name of the tool as specified in the text and Google will give you the home page of the tool.

Agile Manifesto	Manifesto for Agile Software Development http://www.agilemanifesto.org/
Beck	Kent Beck: *Extreme Programming Explained: Embrace Change, 2nd Edition*, Addison-Wesley, 2004.
Beedle	Mike Beedle, Ken Schwaber: *Agile Software Development with Scrum, Prentice Hall*, 2001.
Clark	Mike Clark: *Pragmatic Project Automation*, Pragmatic Bookshelf, 2004.
Cockburn	Alistair Cockburn: *Crystal Clear: A Human-Powered Methodology for Small Teams*, Addison-Wesley, 2004.
Palmer	Stephen R. Palmer, John M. Felsing: *A Practical Guide to the Feature-Driven Development*, Prentice Hall PTR, 2002.
DSDM	DSDM Consortium, Jennifer Stapleton: *DSDM: Business-Focused Development, 2nd Edition*, Addison-Wesley, 2002.
Evans	Eric Evans: *Domain-Driven Design*, Addison-Wesley, 2003.
Fowler	Martin Fowler: *Using an Agile Software Process with Offshore Development* http://www.martinfowler.com/articles/agileOffshore.html
Hunt	Andrew Hunt, David Thomas: *Pragmatic Unit Testing,* Pragmatic Bookshelf, 2003.
Kerth	Norman L. Kerth: *Project Retrospectives: A Handbook for Team Reviews*, Dorset House Publishing, 2001.
Larman1	Craig Larman: *Agile and Iterative Development: A Manager's Guide*, Addison-Wesley, 2003.
Larman2	Craig Larman and Victor R. Basili: *Iterative and Incremental Development: A Brief History*, IEEE Computer, June 2003.
Massol	Vincent Massol: *Junit in Action*, Manning Publications, 2003.
Rainsberger	J.B. Rainsberger: *JUnit Recipes: Practical Methods for Programmer Testing*, Manning Publications, 2004.
Schwaber	Ken Schwaber: *Agile Project Management with Scrum*, Microsoft Press, 2004.
Simmons	Matt Simmons: *Internationally Agile*, published on informIT.com in 2002. http://www.informit.com/articles/article.asp?p=25929
Steindl	Christoph Steindl: *Distributed Agile Project Experience* http://blogs.webahead.ibm.com/pilot/weblogs/page/christoph_steindl@at.ibm.com?catname=Distributed+Agile+Project+Experience
Thomas	David Thomas, Andrew Hunt: *Pragmatic Version Control Using CVS*, Pragmatic Bookshelf, 2003.
Williams	Laurie Williams, Robert Kessler: *Pair Programming Illuminated*, Addison-Wesley, 2002.